# week _4 _EE21B137

March 4, 2023

## 1 Topologically ordered evaluation

[10]:
```
import networkx as nx
```

[11]:
```
def PI(file):
    text_file = open(file,"r")
    data = text_file.read()
    data = data.splitlines()
    text_file.close()
    return data
```

[12]:
```
d_value = {}
```

[21]:
```
def NAND(x,y):
    print(x,y)
    print(d_value[x],d_value[y])
    if d_value[x] == '0' or d_value[y] == '0':return '1'
    else :return '0'
def AND(x,y):
    print(x,y)
    if d_value[x] == '0' or d_value[y] == '0':return '0'
    else :return '1'
def OR(x,y):
    print(x,y)
    if d_value[x] == '0' and d_value[y] == '0':return '0'
    else :return '1'
def NOR(x,y):
    print(x,y)
    if d_value[x] == '0' and d_value[y] == '0':return '1'
    else :return '0'
def XOR(x,y):
    print(x,y)
    if d_value[x] == d_value[y]:return '0'
    else :return '1'
def XNOR(x,y):
    print(x,y)
    if d_value[x] == d_value[y]:return '1'
```

```
    else :return '0'
def NOT(x):
    print(x)
    if d_value[x] == '0':return '1'
    else:return '0'
def INV(x):
    print(x)
    if d_value[x] == '0':return '1'
    else:return '0'
def BUF(x):
    print(x)
    if d_value[x] == '0':return '0'
    else:return '1'
```

[22]:

```
def evaluation(gate,a,node):
    z = node
    if gate == "nand2" :
        d_value[z] = NAND(list(a.predecessors(z))[0],list(a.predecessors(z))[1])
    elif gate == "and2" :
        d_value[z] = AND(list(a.predecessors(z))[0],list(a.predecessors(z))[1])
    elif gate == "or2" :
        d_value[z] = OR(list(a.predecessors(z))[0],list(a.predecessors(z))[1])
    elif gate == "nor2" :
        d_value[z] = NOR(list(a.predecessors(z))[0],list(a.predecessors(z))[1])
    elif gate == "xor2" :
        d_value[z] = XOR(list(a.predecessors(z))[0],list(a.predecessors(z))[1])
    elif gate == "xnor2" :
        d_value[z] = XNOR(list(a.predecessors(z))[0],list(a.predecessors(z))[1])
    elif gate== "not" :
        d_value[z] = NOT(list(a.predecessors(z))[0])
    elif gate == "inv" :
        d_value[z] = INV(list(a.predecessors(z))[0])
    elif gate == "buf" :
        d_value[z] = BUF(list(a.predecessors(z))[0])
    return d_value
```

[ ]:

```
def topo_eval():
    input = PI("c8.inputs")
    print(input)
    input[0]=input[0].split()
    d_gate = {}
    nodes = []
    for p in input[0]:
        d_gate[p] = "PI"
        nodes.append(p)
    net = PI("c8.net")
    print(net)
```

```
    a = nx.DiGraph()
    k = []
    d ={}
    for i in range(0,len(net)):
        net[i]=net[i].split()
        d_gate[net[i][len(net[i])-1]] = net[i][1]
        nodes.append(net[i][len(net[i])-1])
        if net[i][1][len(net[i][1])-1] == "2":
    #        print(net[i])
    #       print(len(net[i][1]))
    #        print(net[i][len(net[i][1])-1])
            k.append((str(net[i][3]),str(net[i][4])))
            k.append((str(net[i][2]),str(net[i][4])))
        else:
            k.append((str(net[i][2]),str(net[i][3])))
    #   print(net[i][1][len(net[i][1])-1])
    #print(nodes)
    #print(k)
    a.add_edges_from(k)
    #print(d_gate)
    nx.set_node_attributes(a,d_gate,name="gateType")
    n2 = list(nx.topological_sort(a))
    print('Nodes in topological order',n2)
    print(a.nodes(data=True))
    #inputs to the PI values
    #print(input)
    for i in range(1,len(input)):
        input[i]=input[i].split()
        for j in range(0,len(input[i])):
            d_value[input[0][j]] = input[i][j]
    #    print(d_value)
        #d_value2.append(d_value)
    #    print(d_value2)
        for z in n2:
    #        print(a.nodes(data=True)[z]["gateType"])
    #        print(z)
            evaluation(a.nodes(data=True)[z]["gateType"],a,z)


        print(d_value)
topo_eval()
%timeit topo_eval()
```

## 2 Event-driven evaluation

[ ]:
```
import networkx as nx
import queue
import time
import numpy as np
#evaluation of gates
def evaluate(gate,input_values):
    if gate == "INV":
        value = int(not input_values[0])
    elif gate == "AND2":
        value = int(all(input_values))
    elif gate == "NAND2":
        value = int(not all(input_values))
    elif gate == "OR2":
        value = int(any(input_values))
    elif gate == "NOR2":
        value = int(not any(input_values))
    elif gate == "XOR2":
        value = int(sum(input_values) % 2)
    elif gate == "XNOR2":
        value = int(not sum(input_values) % 2)
    elif gate == "BUF":
        value = int(input_values[0])
    else:
        raise ValueError("Invalid gate type")
    return value
def statesnew(g,queue,states):
    while queue:
        n=queue.pop(0)
        if (states[n]==1 or states[n]==0):
            continue
        inputstates=[states[i] for i in g.predecessors(n)]
        if None in inputstates:
            queue.append(n)
        else:
            gate=g.nodes[n]["gate_type"]
            k=evaluate(gate.upper(),inputstates)
            states[n]=k
    return
def event_driven_evaluation(net,inputs):
    dic=[]
    f1=open(net,'r')
    fl=f1.readlines()
    g=nx.DiGraph()
    for l in fl:
        p=l.split()
```

```
        try:
            name,gate_type,input1,input2,output=p
            g.add_node(output,gate_type=gate_type)
            g.add_edge(input1,output)
            g.add_edge(input2,output)
        except:
            name,gate_type,input1,output=p
            g.add_node(output,gate_type=gate_type)
            g.add_edge(input1,output)
    f2=open(inputs,'r')
    inputs=f2.readlines()
    length=len(inputs)
    initial_inputs=inputs[0].split()
    inputval=inputs[1].split()
    queue=[]
    for i in g.nodes():
        for s in g.successors(i):
            for t in g.successors(s):
                if t==i:
                    return
                else:
                    continue
    for n in g.nodes():
        queue.append(n)
    states={n: None for n in g.nodes()}
    for i in range(len(initial_inputs)):
        states[initial_inputs[i]]=int(inputval[i])
    statesnew(g,queue,states)
    p={}
    for node in sorted(g.nodes):
        p[node]=states[node]
    dic.append(p)
    for j in range(2,len(inputs)):
        inputval=inputs[j].split()
        for i in range(len(initial_inputs)):
            if (states[initial_inputs[i]] != int(inputval[i])) :
                states[initial_inputs[i]]=int(inputval[i])
                for s in g.successors(initial_inputs[i]):
                    states[s]=None
                    queue.append(s)
                    for k in queue:
                        for t in g.successors(k):
                            states[t]=None
                            queue.append(t)
                            queue=list(set(queue))
        statesnew(g,queue,states)
        q={}
```

```
        for node in sorted(g.nodes):
            q[node]=states[node]
        dic.append(q)
    return dic,length
f1="c8"
f1=f1+'.net'
f2="c8"
f2=f2+'.inputs'
try:
    try:
        k,length=event_driven_evaluation(f1,f2)
        for i in range(length-1):
            print('List of states for inputvector-',str(i+1),':',end='  ')
            print(f"({k[i]})")
            print('\n')
    except TypeError:
        print("The given circuit is not acyclic i.e the given circuit contains␣
 ↪loops")
    print("time elapsed for the compilation is")
    %timeit event_driven_evaluation(f1,f2)
except FileNotFoundError:
    print("Error: Files named ",f1,"and",f2,"not found:")
```

## 3 comparision of time elapsed

[ ]:
```
%timeit topo_eval()
%timeit event_driven_evaluation(f1,f2)
```

## 4 Discussion about results

- As the *event driven evaluation* approach takes less number of values to evaluate when compared to the *Topologically ordered evaluation* approach ,here **event driven evaluation** approach takes less time generally.
- The comparision of the time taken for both the functions to run is done by using the above program.
- The *event-driven evaluation* approach takes less time complexity than the *topologically ordered evaluation* approach. In terms of the number of nets, the time complexity of the *event-driven evaluation* approach is O(NK), where N is the number of nets and K is the number of inputs. This is because for each input combination, the event-driven approach only updates the values of the affected gates, while the *topologically ordered evaluation* approach recomputes the values of all gates for each input combination, resulting in a time complexity of O(N(2^K)). Therefore, the **event-driven evaluation** approach is more efficient for larger circuits with many gates.
- In general, **event-driven evaluation** can be more efficient than **topologically ordered evaluation** when there are only a few inputs changing at each time step, whereas topological evaluation can be more efficient when many inputs are changing at each time step. However,

it's difficult to make a definitive statement about which approach is faster without more information about the specific circuit and input values being used.